ISSN: 2745-3673
http://www.Rcsj.ir

مجلـه‌ی علمی پژوهش در علوم رایانه (سال سوم)
شماره ۱۱ / پاییز ۱۳۹۷ / ص ۴۵- ۲۶

بهبود کیفیت آزمون در نرم‌افزارهای حساس به ایمنی با استفاده از آزمون جهش

احسان خراطی
دانشگاه آزاد اسلامی واحد اراک، دانشکده فنی و مهندسی، گروه کامپیوتر نرم‌افزار.

نام نویسنده مسئول:
احسان خراطی

چکیده

آزمون یک راه اولیه برای اطمینان از نرم‌افزار، در سیستم‌های حساس به ایمنی است. برای این منظور باید فرآیند آزمون را به اندازه کافی اجرا کرده و اثبات کرد که تمام سطوح آزمون پوشش می‌شوند. سطوح پوشش اغلب یا توصیه شده‌اند یا براساس استانداردهای امنیتی و دستورالعمل‌های صنعتی وضع شده‌اند. آزمون جهش، یک روش مکمل یا جایگزین برای اندازه‌گیری کارایی آزمون است. اما به‌طور گسترده‌ایی در نرم‌افزارهای صنعتی و یا حساس به ایمنی بکار نمی‌رود. در این مقاله، آزمون جهش در سیستم‌های نرم‌افزاری هوایی که حساس به ایمنی هستند با پوشش اکثر نیازمندی‌ها و با استفاده از عملگرهای سطح بالا در زبان‌های C و Ada انجام می‌شود. همچنین در این مقاله موثرترین انواع جهش و منشا عدم موفقیت‌ها شناسایی شده و سپس تحلیل و بررسی شده و تا بتوان رابطه‌ای بین مشخصات برنامه نظیر تعداد خطوط کد و پیچیدگی مداری و پایداری جهش و جهش‌های زنده مانده و خطاهای نهفته یافت.

نتایج نشان می‌دهد که آزمون جهش می‌تواند نسبت به تحلیل پوشش ساختاری سنتی و بررسی دستی موثرتر باشد و سبب بهبود فرآیند برنامه‌نویسی و تعریف نیازمندی‌ها گردد. همچنین در زمینه‌های صنعتی نظیر نرم‌افزارهای هوایی نیز می‌توان از آزمون جهش استفاده کرد.

واژگان کلیدی: آزمون جهش، حساس به ایمنی، اطمینان، کارایی.

مجلـه‌ی علمی پژوهش در علوم رایانه ، شماره ۱۱ ، پاییز ۱۳۹۷، ص ۴۵- ۲۶

ISSN: 2745-3673
http://www.Rcsj.ir

مقدمه

آزمون یک فعالیت ضروری برای تایید و ارزیابی نرم‌افزارهای حساس به ایمنی بوده و با استفاده از پوشش نیازمندی و ساختارهای کد می‌توان سطح اطمینان در محصول نهایی را افزایش داد. در [11] پیشنهاد شده که در پروژه‌های نرم‌افزاری حساس به ایمنی ۶۰ تا ۷۰ درصد از کل تلاش خود را صرف تایید و ارزیابی نیازمندی‌ها معطوف نماییم. در [29] رابطه بین هزینه و کارایی آزمون را تحلیل کرده و روش‌های جدیدی را برای آزمون که قبلا به علت محدودیت‌های محیطی غیرممکن بود را معرفی کرده که یکی از این روش‌ها آزمون جهش است. در [13] آزمون جهش را با استفاده از جایگزین‌های دستوری در نرم‌افزار و تکرار فاز اجرایی آزمون و اصلاح کدها برای افزایش اطمینان و توسعه نمونه آزمایشی در دهه ۷۰ پیشنهاد و معرفی کرده است. هرچه نمونه‌های آزمایشی اصلی در کدهای اصلاح شده بیشتر تشخیص داده شوند، کیفیت آزمون جهش بیشتر خواهد شد. با وجود مزیت‌های بالقوه آزمون جهش به‌طور گسترده در صنعت و امور حساس به ایمنی استفاده نشده است زیرا به کار و منابع محاسباتی زیادی نیاز دارد. اما با افزایش قدرت پردازشی و ابزارهای پشتیبانی و تحقیقات بیشتر، آزمون جهش با داشتن مزیت سادگی فرآیندهای جهش، می‌تواند در امور صنعتی حساس به ایمنی نیز استفاده شود.

هدف این مقاله، ارزیابی تجربی آزمون جهش درون یک محیط حساس به ایمنی با استفاده از ۲ سیستم نرم‌افزار هوایی در دنیای واقعی است. این سیستم‌ها با استفاده از زبان‌های Ada وC توسعه داده شده‌اند که قابلیت توسعه پوشش دستورالعمل و تصمیم یا MC/DC را دارند. ما در این مقاله سعی کردیم بیشتر روی رابطه بین پیچیدگی کد و نرخ زنده ماندن جهش تمرکز کرده که برای مطابقت با اهداف DO-178B و حفظ پوشش سطوح مورد نیاز سبب طولانی‌تر شدن آزمون گردید. همچنین آزمون جهش و ایجاد خطا را در زبان Ada به‌صورت دستی و قضاوت ذهنی انجام دادیم که سبب تشخیص و کاهش جهش‌های معادل شد. درحال حاضر شواهد تجربی کمی در رابطه با تاثیر آزمون جهش در بهبود ارزیابی نرم‌افزارهای حساس به ایمنی وجود دارد و هیچ انگیزه‌ای برای استفاده از آزمون جهش در زمینه هوا فضا برای مهندسان نرم‌افزار وجود ندارد زیرا اکثر دستورالعمل‌های امنیتی نرم‌افزاری نیاز به کاربرد آزمون جهش ندارند. به همین منظور، سعی این مقاله دنبال کردن اهداف زیر است:

۱- تعریف یک زیرمجموعه موثر از انواع جهش‌های کاربردی برای سیستم‌های نرم‌افزاری حساس به ایمنی با استفاده از SPARK Ada [6] و MLSRA C [28]

۲- یافتن و طبقه‌بندی ریشه همه شکست‌ها در نمونه‌های آزمایشی.

۳- یافتن رابطه‌ای بین مشخصات برنامه نظیر اندازه یا تعداد خطوط کد برنامه و جهش‌ها و خطاهای نهفته یا زنده مانده در داده آزمون.

۴- مقایسه آزمون جهش و آزمون دستی.

سازماندهی این مقاله به این ترتیب است که بخش ۲ کارهای صورت گرفته در این زمینه را بررسی کرده و بخش ۳ شامل مفاهیم اصلی و اساسی در مورد نرم‌افزارهای حوزه هوافضا و معرفی آزمون جهش است. بخش ۴ شامل تعریف حوزه و فرآیند جهش است و بخش ۵ نتایج را ارزیابی می‌کند و درنهایت بخش ۶ شامل نتیجه‌گیری و بحث است.

۱- کارهای مرتبط

در [21] نشان می‌دهد که مطالعه روی آزمون جهش بیش از ۳ دهه است و از سال ۲۰۰۶ به بعد مقاله‌های بیشتری براساس شواهد تجربی و برنامه‌های کاربردی نسبت به تئوری جهش منتشر شده است و آزمون جهش از تحقیقات پایه درحال حرکت به سوی مفاهیم صنعتی پیشرفته است. در [12] نشان داده است که آزمون جهش می‌تواند در پروژه‌های نرم‌افزاری حساس به ایمنی نظیر هسته‌ای و عمرانی توسعه یابد. مچنین این مقاله رابطه بین پیچیدگی مداری و تولید جهش و زنده ماندن جهش‌ها را بررسی می‌کند. در [40] رابطه بین پیچیدگی کد و آزمون جهش را بررسی کرده و نشان می‌دهد که هرچه پیچیدگی کدآیتم افزایش یابد، احتمال خطاهای کد نیز افزایش می‌یابد که مستلزم توسعه داده آزمون علیه خطاهاست. در [25] رابطه بین پیچیدگی کد و قابلیت اطمینان بررسی شده است. در [36] و [18] استفاده از پیچیدگی مداری به‌عنوان یک پارامتر درمقابل اندازه پیچیدگی کد مورد انتقاد قرار گرفته است و لذا تمام روابط بین نرخ زنده ماندن جهش و پیچیدگی مداری را نیز مورد انتقاد قرار می‌دهد. در [30] از پوشش دستورالعمل برای مقایسه ضعیف بودن آزمون استفاده کرده اما پوشش دستورالعمل، خطاهایی را که در شاخه و شرط رخ می‌دهند را شناسایی نمی‌کند. در [9] از مجموعه‌ای آزمون براساس ساختارهای MC/DC برای تشخیص انواع خطاها استفاده می‌کند اما تضمینی را برای شناسایی همه خطاهای MC/DC نمی‌دهند. همچنین نشان می‌دهد که با افزایش پوشش شرط و دستورالعمل در DO-178B و تعداد آزمون‌ها، احتمال تشخیص خطاهای بیشتری وجود دارد. همچنین برای بررسی توانایی و اثربخشی فرم‌های مختلف MC/DC از یک سری آزمون جهش استفاده کرده است و نتیجه

ISSN: 2745-3673
http://www.Rcsj.ir

می‌گیرد که آزمون جهش می‌تواند پوشش‌هایی را فراهم کند که MC/DC نمی‌تواند و نتیجه دیگر اینکه وقتی پوشش آزمون شامل MC/DC می‌شود، اثربخشی جهش‌های منطقی کاسته می‌شود. در [3] یکسری از نتایج ارزیابی تجربی آزمون جهش در برنامه C و معیار پوشش را نشان داده که برنامه این آزمون توسط آژانس فضایی اروپا بوده و چهار معیار پوشش P-Use، C-Use، Decision، Block را مقایسه می‌کند. این مقاله نشان می‌دهد که چگونه آزمون جهش می‌تواند به ارزیابی هزینه‌های اثربخشی معیار پوشش آزمون کمک کند و ارتباط بین معیارهای اندازه آزمون و نرخ شناسایی جهش‌ها را نشان می‌دهد. در [10] ایجاد خطا به‌صورت دستی و قضاوت ذهنی انجام شده است و نشان داده است که ایجاد و شناسایی این خطاها نسبت به خطاهایی که به‌صورت خودکار تولید می‌شوند متفاوت و سخت‌تر است. مشکلات ایجاد خطا به‌صورت دستی در آزمون جهش شامل پس‌رفت آزمون، محدود شدن مطالعه‌های تجربی برای نرم‌افزارهای حساس به ایمنی بوده که زبان Ada فاقد پشتیبانی خودکار برای تولید جهش است.

۲- نرم‌افزارهای حساس به ایمنی در دامنه هوافضا

شکست در نرم‌افزارهای حساس به ایمنی سبب آسیب‌های انسانی یا محیطی می‌شود [26]. نرم‌افزار DO-178B شامل دستوالعمل‌های اولیه برای تایید نرم‌افزار هوایی حساس به ایمنی در حوزه هوافضا با سطح اطمینان و امنیت بالا مطابق با الزامات صلاحیت پرواز است [35]. این نرم‌افزار شرایط امنیتی را به ۵ دسته، فاجعه‌بار، خطرناک یا عمده شدید، عمده، جزیی و بی‌تاثیر تقسیم کرده و سطوح اطمینان را به ۵ سطح از A یا بالا که مربوط به سیستم‌های بلادرنگ است تا E یا شرایط شکست است تقسیم می‌کند [37,38]. به‌عنوان مثال، برای بدست آوردن سطح اطمینان C در نرم‌افزار، لازم است حداقل به ۵۷ هدف دست بیابیم و برای بدست آوردن سطح اطمینان A در نرم‌افزار نیاز به ۹ هدف بیشتر است. چون نمی‌توان یک آزمون کامل را برای یک سیستم نرم‌افزاری پیچیده انجام داد، نیاز به یک شاخص تایید برای ارزیابی کیفیت آزمون نرم‌افزار است [7, 23]. دو تا از این شاخص‌ها مربوط به الزامات پوشش و ساختار کد هستند. به‌عنوان مثال، برای رسیدن به سطح C نرم‌افزار، نیاز به پوشش دستورالعمل در ساختار برنامه‌نویسی است [35] و برای رسیدن به سطح A نرم‌افزار، نیاز به پوشش دستورالعمل و شرط و تصمیم در ساختار برنامه‌نویسی است [14].در نرم‌افزار DO-178B برای آگاهی از اهداف و کفایت آزمون از تحلیل دستی و بازنگری استفاده می‌شود. مشکل تحلیل دستی، اندازه‌گیری کیفیت است. آزمون جهش یک فرآیند قابل تکرار بوده و برای اندازه‌گیری کیفیت مفید است و می‌تواند نابرابری‌ها در مجموعه آزمون و داده آزمون را شناسایی کند.

همچنین آزمون جهش با استفاده از جایگزینی ساختارهای ساده کد و عملوندهایی با ساختار مصنوعی مجاز برای شبیه‌سازی سناریوهای عدم موفقیت استفاده شود. [13]

برنامه‌های جهش‌یافته می‌توانند علیه داده آزمون اصلی مجددا اجرا شوند که درنتیجه می‌توان مشخص کرد که آیا یک نمونه داده آزمون می‌تواند جهش را ازبین ببرد یا نه. اگر جهش‌ها توسط داده آزمون‌ها کشته نشوند، بیانگر این است که داده آزمون‌ها کارآمد نبوده و باید بهبود یابند. این فرآیند آزمون مجددا اجرا شده تا همه جهش‌های تولید شده، توسط داده آزمون‌ها کشته شوند. لذا اثربخشی آزمون جهش وابسته به داده آزمون‌ها و شناسایی دسته‌بندی‌های متفاوت از خطاهای برنامه‌نویسی است. به‌عنوان مثال، یک سری از داده آزمون‌ها می‌توانند معیار پوشش مناسبی داشته باشد اما ممکن است در شناسایی انواع خاصی از خطاهای برنامه‌نویسی موفق نباشند لذا تحلیل پوششی ساختار و آزمون جهش می‌توانند مکمل یکدیگر باشند. اما در تحلیل پوشش ساختاری دستورالعمل یا MD/DC تا یک حدی می‌توان با داده آزمون‌ها عمل خطایابی را انجام داد. در [13] آزمون جهش را براساس ۲ فرض اصلی در انواع خطاها که به‌طور معمول در نرم‌افزار اتفاق می‌افتد، توجیه می‌کند. فرض اول آن است که برنامه‌نویسان باکفایت هستند و برنامه‌های آنها به‌طور معمول به برنامه خواسته شده نزدیک است. فرض دوم آن است که هر خطای پیچیده نتیجه یک یا چند شکست ساده درون نرم‌افزار است. بنابراین با جایگزینی‌های ساده می‌توان انواع خطاهای معمول که توسط توسعه‌دهنده‌ها ایجاد شده‌اند را یافت و با تقویت داده آزمون‌ها، جهش‌هایی که شامل این نوع خطاها هستند را کشت. درنتیجه از رخداد شکست‌های پیچیده در نرم‌افزار جلوگیری می‌شود.

تحقیقات در زمینه آزمون جهش و اطمینان از اثربخشی آن، اغلب توسط Offutt و Untch[34] انجام شده که شامل Do Fewer که به معنی روش‌هایی برای کاهش تعداد جهش‌ها را در عملوندها و Do Faster که به معنی روش‌هایی برای اجرایی سریع‌تر آزمون و Do Smarter که به معنی روش‌هایی برای بکارگیری جهش در نقاط و سطوح متفاوت است. موارد بالا در محل فرآیند جهش که در فاز تولید یا اجرای داده آزمون یا تحلیل نتایج است باهم تفاوت دارند. به‌طورکلی فرآیند جهش شامل به‌کارگیری عملگر جهش است. [19]

۳- فرآیند جهش:

دو سیستم نرم‌افزاری توسعه داده شده در Aero Engine Controls(AEC) به نام‌های Roll-Royce و Goodrich برای این مقاله انتخاب شدند. این سیستم‌ها، عملکردهای متفاوتی را به‌عنوان قسمتی از Full Authority Digital Engine(FADEC) اجرا می‌کنند و با

ISSN: 2745-3673
http://www.Rcsj.ir

استفاده از زبان‌های برنامه‌نویسی مختلف تا سطوح متفاوت تایید توسعه داده شده‌اند. هر دوی این سیستم‌ها تا سطح آزمون پوشش الزامی توسط مسوولان مربوطه مجاز، تایید شده‌اند. سیستم Roll-Royce یک برنامه کاربردی کنترل موتور ساختارهای هوایی است که در SPARK Ada و DO-178B تا سطح A توسعه داده شده است و برای بدست آوردن شاخص MC/DC تحت بررسی است. این سیستم به‌عنوان پروژه A در این مقاله تعیین شده است. سیستم Goodrich نیز یک برنامه کاربردی برای نظارت و تشخیص روی موتور هواپیما بوده که در MISRA C و DO-178B تا سطح C توسعه داده شده است و برای کسب شاخص پوشش سطح دستورالعمل تحت بررسی است. این سیستم به‌عنوان پروژه C در این مقاله تعیین شده است. به دلایل محرمانه و تجاری بودن کدآیتم‌های نمونه‌گذاری شده آنها را از پروژه A با A1 تا A25 و از پروژه C با C1 تا C22 و کدآیتم‌های اضافی از پروژه C را با R1 تا R3 نشان می‌دهیم.

در این مقاله کدآیتم، یک برنامه نرم‌افزاری بوده که حاوی تقریبا ۲۰ خط از کدهای دستورالعمل یا LOC است و نمادی از طراحی ماژولاریتی است و هر کدآیتم، عملکرد، اندازه و پیچیدگی خود را دارد. انتخاب کدآیتم‌ها به‌منظور به حداکثر رساندن تعداد انواع جهش‌های مناسب، براساس عملکردهای متنوع و سطوح پیچیدگی نظیر پیچیدگی مداری است [40]. انتخاب زبان برنامه‌نویسی، عامل بسیار مهمی در تعداد جهش‌های مناسب در یک کدمنبع است. به‌عنوان مثال زبان Ada که یک زبان قوی از نظر نوع است نیاز به جایگزینی‌های کمتری نسبت به زبان C که از نظر نوع سریع‌تر است دارد. دستورات جهش ممکن است محدودیت‌هایی در استانداردها و روش‌هایی نظیر MISRA داشته باشند که این مسأله می‌تواند مجددا موجب کاهش و جایگزین‌های مجاز در خلال آزمون جهش شود. درحال حاضر هیچ مجموعه ابزاری برای ایجاد جهش در زبان Ada وجود ندارد لذا فرآیند تولید جهش در پروژه A به‌صورت کاملا دستی انجام شده و برای پروژه C، فعالیت‌های جهشی توسط مجموعه ابزار MILO انجام شده است [20]. چون ابزارهای جهش C در دسترس است لذا می‌توان درصد بزرگتری از نرم‌افزار را تحلیل کرد. عملگرهای سطح پایین در هر سیستم نرم‌افزاری و زبان برنامه‌نویسی یکسان هستند.

در این مقاله، ابتدا آزمون جهش را روی یک نمونه از کدآیتم‌ها در پروژه C اجرا کرده که نتایج آن سبب شناسایی مجموعه‌ای از انواع جهش‌های موثر مربوط به SUT شد. سپس در مرحله بعدی این مجموعه موثر از عملگرهای جهش یافته را روی پروژه A اجرا کردیم و نتایج آنها را مقایسه کردیم که یک زیرمجموعه موثر از عملگرهای جهش یافته در زبان Ada را بدست آوردیم.

در [34] فقط روی جهش‌های محدودی تمرکز شده است اما در محیط آزمون ما از ابزارهای اختصاصی استفاده شده که به آسانی قابلیت انتقال به ابزارهای آزمون جهش دردسترس دیگر را دارند. فاز اجرایی آزمون مقاله ما، در یک محیط شبیه‌سازی آزمون بوده که قابلیت ایجاد و اجرای جهش‌ها به دفعات طولانی و به‌صورت موازی را دارد لذا سربار زمان اجرایی کاهش می‌یابد.

۳-۱- جهش‌های انتخابی زبان‌های C و Ada

تمام دستورالعمل‌های یک کدآیتم باید بررسی شده و عملگرهای جهش مناسب برای آن شناسایی گردد و داده آزمون متناسب با آن را یافته تا از اجرای آن مطمئن شده و موفق به پوشش دستورالعمل یا MC/DC گردیم. فعالیت جهش برای کدآیتم‌های Ada به‌صورت دستی و برای کدآیتم‌های C از مجموعه ابزار MLLU استفاده می‌کنیم که دلیل آن توانایی عملوندها و جایگزینی عبارت‌ها است. این ابزار چون در جهش‌هایی با مرتبه بالاتر بکار رفته‌اند توسعه یافته‌اند و برای جهش‌های مرتبه بالا در زبان C امکاناتی را فراهم کرده‌اند. هر عملگر جهش می‌تواند کاربردهای مختلفی داشته باشد که سبب افزایش پیچیدگی ارزیابی موفقیت هر جهش می‌شود. جهش‌های زنده می‌توانند داده آزمون‌ها را بهبود داده تا با تکرار آزمون‌ها، جهش‌های زنده بالاخره کشته شود. در [31] عملگرهای جهش شامل دستورات عبارتی و پوششی و وظیفه‌ای است که با مجموعه MILU می‌توان جهش‌های عبارتی و پوششی را پوشش داد اما جهش‌های وظیفه‌ای غیرقابل پوشش هستند.

برای هر نوع جهش، یک مجموعه از جایگزینی‌های ازپیش تعریف شده وجود دارد که همه آنها در طی آزمون جهش C استفاده شده‌اند اما چون قواعد نحوی زبان Ada متنوع است، شامل جهش‌ها و دامنه جایگزینی بیشتری است. در [1]، برای زبان C تعداد ۷۷ عملگر جهش را شناسایی کرده که برای ۴۸ عدد از آنها MILU بکار رفته است. در [31] نیز برای زبان Ada تعداد ۶۵ عملگر جهش را پیشنهاد کرده که بسیار شبیه به مجموعه زبان C است. چون اکثر ساختارهای کد یکسان‌اند، لذا اغلب جهش‌ها برای سیستم‌ها یکسان هستند.

در این مقاله، از ۴۶ عملگر جهش پایه برای هر دو مجموعه فوق استفاده شده و از قرارداد نامگذاری پیشنهادی MILU استفاده شده است. این جهش‌ها شامل:

۱- جایگزینی‌های ثابت که توسط یک سیستم جهش مبتنی برقاعده خودکار انجام می‌شوند.

۲- جهش‌های نفی که توسط MILU ارایه نشده‌اند و رفتاری مشابه عملگرها را دارند.

۳- جهش‌های ساختمان خاص که در نرم‌افزارهای حساس به ایمنی استفاده شده‌اند و شامل تعیین بن‌بست و بررسی دستورات while، break و goto است.

ISSN: 2745-3673
http://www.Rcsj.ir

۴- جهش‌های نوع داده که فقط شامل اشاره‌گرها نمی‌شوند.

دو تا از جهش‌های سطح دستورالعمل که شامل افتادن در دام دستورالعمل اجرایی یا STRP و افتادن در دام شرط یا STRI است موجب توقف آزمایش شدند و در این مقاله درنظر گرفته نشده‌اند.

۳-۲- جهش‌های معادل:

جهش‌های معادل جهش‌هایی هستند که هرگز با داده آزمون‌ها کشته نمی‌شوند و رفتارشان بسیار مشابه با برنامه اصلی است و لذا هیچ ارزشی را به فرآیند آزمون اضافه نمی‌کنند [21]. در [39] نشان داده شده که اغلب جهش‌های معادل در جایگزین‌های منطقی و محاسباتی و رابطه‌ای وجود دارد. هنگام شناسایی جهش‌های زنده لازم است که تک‌تک آنها به‌طور دستی بازبینی شوند تا معادل بودن آنها مشخص گردد. بازنگری جهش‌های زنده اغلب به‌صورت دستی بوده که مستلزم صرف سربار زمانی بوده و همچنین بازنگری جهش‌های زنده به‌صورت خودکار بسیار مشکل است. هرچند با خودکارسازی تولید جهش، فرآیند جهش بهبود می‌یابد، اما احتمال ایجاد جهش‌های معادل نیز افزایش می‌یابد. تحلیل جهش‌های زنده و یافتن جهش‌های معادل، عوامل مهمی در محاسبه امتیاز جهش هستند. زیرا امتیاز جهش برابر است با تعداد جهش‌های کشته شده تقسیم بر تعداد جهش‌های غیرمعادل [21].

عوامل اصلی جهش‌های معادل اغلب گرفتن داده آزمون یکسان برای ورودی و خروجی هر طرف منطقی و یا روابط قیاسی و جهش مقایسه‌ها و عدم شناسایی پوشش‌ها است که سبب کشته نشدن جهش‌ها خواهد شد. بنابراین تعیین هم‌ارزی جهش‌ها نیاز به یک بازنگری وسیع از کدها داشته و فقط یک تحلیل جزئی محلی کافی نیست. لذا در این مقاله یک بازنگری دستی از جهش‌های معادل انجام شده است. بهبود شناسایی جهش‌های معادل و پرهیز از آنها و تحلیل خودکارسازی آنها یک زمینه بسیار مهم تحقیقاتی در آینده می‌تواند باشد که چندین روش براساس نظیر برش برنامه [15] بهینه‌سازی کامپایلر [4]، حل محدودیت [33] برای معضل جهش‌های معادل پیشنهاد شده است.

با این وجود، اغلب از تحلیل دستی جهش‌های معادل در پروژه‌های A و C استفاده می‌شود تا با جایگزینی جهش، جهش معادل را همانندسازی کرده و یا تعداد جهش‌های معادل و تعداد دفعات اجرای آزمون را کاهش داد.

در این مقاله جهش‌های زنده به‌صورت زیر دسته‌بندی شده‌اند:

۱- جهش‌هایی که معادل هستند و هرگز کشته نمی‌شوند.

۲- جهش‌هایی که معادل هستند اما علت آن عدم تخصیص اولیه در یک متغیر محلی است.

۳- جهش‌هایی که معادل نیستند و خروجی برنامه اصلی با برنامه جهش یافته متفاوت است و جهش‌ها باید توسط داده آزمون‌ها کشته شوند، ولی هنوز کشته نشده‌اند.

دسته دوم جهش‌های زنده، می‌توانند به‌علت مختصرنویسی در زبان C در اجرای عبارات محاساباتی و یا انتساب در یک دستور بوجود بیایند. به‌عنوان مثال، اگر دستورالعمل مقداردهی اولیه a=1 را به a+=1 جهش دهیم سبب عدم مقداردهی اولیه شده و متغیر محلی a حافظه را به‌صورت پویا به خود اختصاص داده و حالت اولیه آن را نمی‌توان صفر فرض کرد و لذا این متغیر که قبلا تاثیری روی خروجی نداشت اکنون تاثیرگذار خواهد بود. بنابراین برای یافتن خطا نیاز به تحلیل پیچیده دستی است زیرا عامل اصلی خطاها، دستورالعمل‌ها سازگار در زبان برنامه‌نویسی است. اما اگر این خطاها با روش‌های طراحی ناسازگار باشند، در هنگام طراحی تشخیص داده می‌شوند و کار راحت‌تر خواهد بود. متغیرهای موقت از پشته استفاده می‌کنند و لذا استفاده از این متغیرها، قبل از راه‌اندازی صحیح آنها سبب پیچیده شدن تحلیل جهش برای تعیین هم‌ارزی‌ها خواهد شد.

در زبان C، با استفاده از دستورالعمل‌های MISRA C از این نوع نحو جلوگیری می‌شود و در زبان SPARK Ada نیز چنین عملیاتی مجاز نیست. جهش‌هایی که در فاز اولیه آزمون کشته نشدند، لازم است مجددا مورد بازبینی قرار گیرند که این بازبینی به‌منظور تحلیل رفتارهای معادل است. در این مقاله مهمترین عوامل ایجاد جهش‌های معادل را به‌صورت زیر طبقه‌بندی کرده‌ایم:

۱- مقادیر اولیه برای متغیرهای موقتی، غیرقابل پیشبینی‌اند زیرا تخصیص حافظه در زمان اجرا انجام می‌شود.

۲- متغیرهای محلی توسط سایر برنامه‌ها غیرقابل دسترسی‌اند و لذا نمی‌توان به‌طور محلی آنها را آزمایش کرد.

۳- رفتار متغیرهای محلی را می‌توان در خروجی برنامه آزمون کرد.

۴- در جایی‌که مقادیر اولیه متغیرهای محلی مشخص نیست، احتمال یک جهش معادل می‌تواند باشد.

۳-۳- ارزیابی روش‌ها آزمون جهش در دو پروژه C و Ada

از ۴۶ عملگر جهش موجود در MZLU، بیش از ۲۲ مورد آن برای کدآیتم C بکار رفتند. نتایج به‌صورت زیر خلاصه شده است:

ISSN: 2745-3673
http://www.Rcsj.ir

در MZLU به تعداد ۳۱۴۹ جهش تولید کرده که ۵۹۴ یا ۱۸ درصد آنها کشته شدند. جهش‌های کشته شده از لحاظ نحوی غیرمجاز بودند و کامپایلر به‌راحتی آنها را تشخیص داده است و ۲۵۵۵ جهش، زنده مانند. با گردآوری آنها و اجرا مجدد آزمون با داده آزمون، فقط ۳۲۳ آنها زنده ماندند. سپس بررسی کردیم که ۱۹۹تای جهش‌های زنده، معادل هستند که ۴۵تای آنها به‌دلیل عدم مقداردهی اولیه به متغیرهای محلی شناسایی شدند و خروجی برنامه را تحت تاثیر قرار می‌دادند. ۷۹ جهش زنده باقیمانده را طبقه‌بندی کردیم تا تعیین کنیم که کدام یک از این ۴۶ عملگر جهش در شناسایی نواقص آزمون موثر بودند.

در جدول ۱، ۲۲ کدآیتم نمونه ساده شده به زبان C را نشان می‌دهد که جهش‌های زنده بین ۱۱ کدآیتم توزیع شده‌اند.

به‌عنوان مثال، در مورد C8 امتیاز کفایت جهش به‌طور قابل ملاحظه‌ای زیر ۱۰۰ درصد است. کدآیتم C8 نسبتا ساده و کوچک است که مجموعه داده آزمون آن شامل یک خطای بنیادی است و به‌علت امتیاز پایین جهش آن، به‌عنوان یک خطای عمده ظاهر می‌شود که دلیل آن اغلب مرتبط با اندازه کدآیتم است. اگر کدآیتم، پیچیده یا بزرگ باشد به علت تعداد جهش‌های زیادی که می‌توانند تولید شوند، تاثیر بسیار اندکی روی امتیاز جهش خواهد داشت.

از جدول ۱ می‌توان نتیجه گرفت که هرچه اندازه و پیچیدگی‌های کدآیتم بیشتر باشد، رخداد نواقص آزمون بیشتر خواهد شد. در بخش‌های بعدی، این رابطه و دلایل چگونگی این ارتباط بین پیچیدگی مداری و پیچیدگی کد و اندازه کد و راه‌های بهبود کیفیت آزمون مورد بحث قرار می‌گیرد.

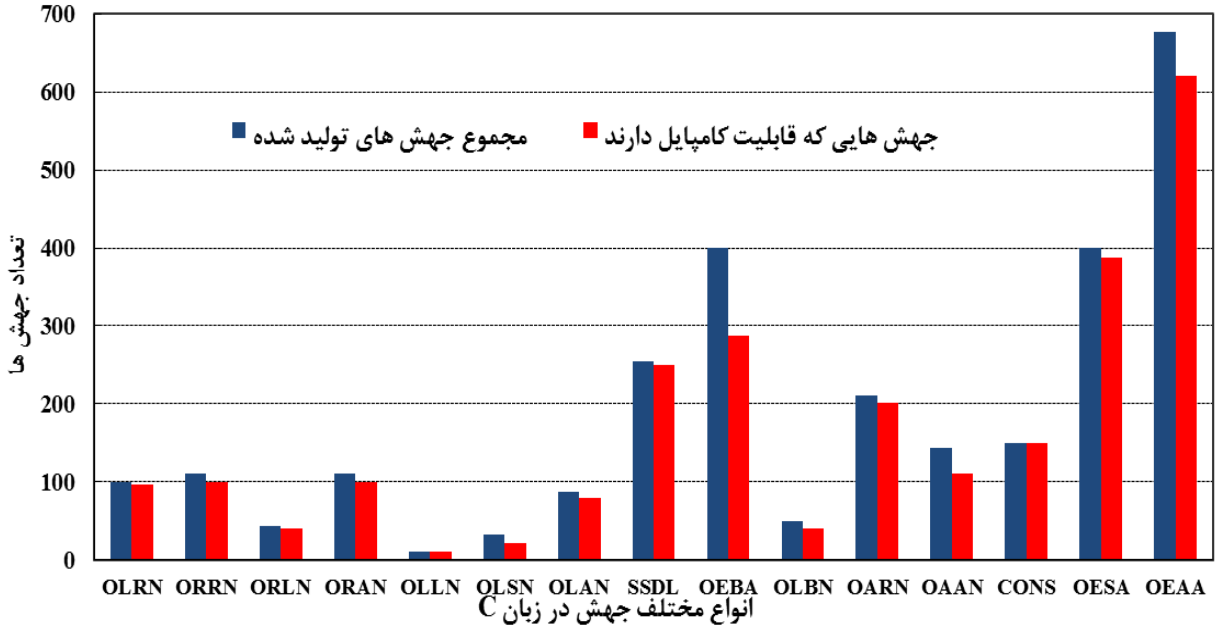
۲۵ عملگر از ۴۶ عملگر MLLU در مورد کدآیتم‌های آزمون شده قابل کاربرد و موثر بودند. تعداد جهش‌های تولید شده از ۱۶ مورد برای OLLN (عملگر منطقی جهش) تا ۶۶۵ مورد برای OEAA (انتساب ریاضی) بود.

شکل ۱، توزیع جهش‌های مرتبط را نشان داده و شکل ۲، تعداد کل جهش‌های معادل و زنده مانده در آزمون را نشان می‌دهد.

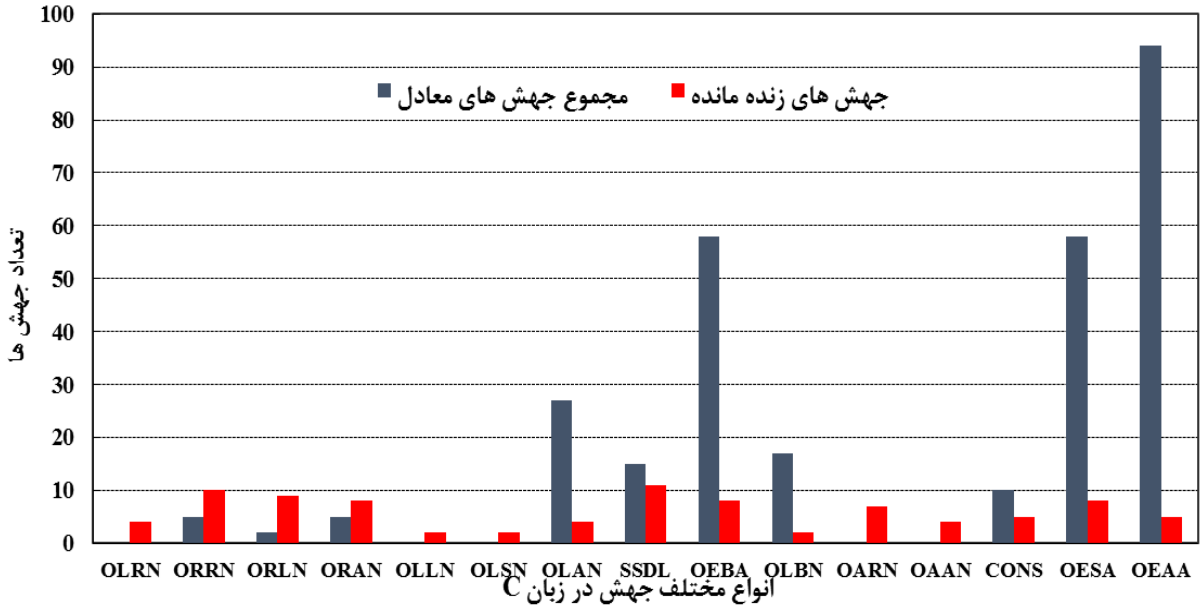
جدول ۱: شایستگی جهش از هر کدآیتم پروژه C

کد منبع			فاز تولید		اجرای آزمون		تحلیل		
آیتم کد	تعداد خطوط کد	پیچیدگی مداری	تعداد کل جهش‌های تولیدشده	جهش‌های نادرست نحوی	جهش‌های کشته نشده	مجموع جهش‌های مجددا آزمون شده	جهش‌های معادل	جهش‌های زنده مانده	امتیاز جهش
C1	15	3	77	11	23	66	23	0	100.00
C2	12	2	142	60	2	82	2	0	100.00
C3	21	3	118	2	14	116	14	0	100.00
C4	8	3	78	2	27	76	25	2	96.08
C5	24	3	111	14	16	97	16	0	100.00
C6	16	2	208	83	14	125	13	1	99.11
C7	7	1	86	85	0	1	0	0	100.00
C8	6	3	101	31	45	70	2	43	36.76
C9	10	2	52	18	2	34	2	0	100.00
C10	33	6	139	8	14	131	13	1	99.15
C11	44	7	217	4	32	213	21	11	94.27
C12	31	5	228	8	5	220	5	0	100.00
C13	6	2	35	1	1	34	1	0	100.00
C14	26	4	223	5	11	218	10	1	99.52
C15	32	6	238	15	39	223	35	4	97.87
C16	46	7	337	81	36	256	30	6	97.35
C17	15	3	777	0	16	77	16	0	100.00
C18	12	1	286	120	0	166	0	0	100.00
C19	42	11	159	0	6	159	4	2	98.71
C20	3	1	48	18	5	30	0	5	83.33
C21	18	5	186	28	15	158	12	3	97.95
C22	8	1	3	0	0	3	0	0	100.00

ISSN: 2745-3673
http://www.Rcsj.ir



شکل ۱: انواع جهش‌های منطبق بر زبان C



شکل ۲: جهش‌های زنده و معادل در پروژه C

از ۲۵ عدد انواع معتبر جهش منطبق بر کد، ۱۵ نوع از آنها در اولین دور از اجرای آزمون زنده ماندند و به‌عنوان نماینده‌های معتبری از نواقص آزمون تلقی می‌شوند. این جهش‌ها در جدول ۲ به‌ترتیب درصد نرخ موفقیت لیست شده‌اند و به‌عنوان یک نسبت از جهش‌های درج شده از آن نوع محسوب می‌شوند.

جدول ۲: نرخ زنده ماندن جهش در زبان C

جهش	ویژگی	نرخ زنده ماندن
OLRN	عملگر منطقی ازطریق عملگر رابطه‌ای	۱۰,۴
ORRN	عملگر رابطه‌ای	۹,۶

ISSN: 2745-3673
http://www.Rcsj.ir

ORLN	عملگر رابطه‌ای ازطریق عملگر منطقی	۸,۷
ORAN	عملگر رابطه‌ای ازطریق عملگر ریاضی	۸,۶
OLLN	عملگر منطقی	۶,۳
OLSN	عملگر منطقی ازطریق عملگر شیفت	۶,۳
OLAN	عملگر منطقی ازطریق عملگر ریاضی	۵,۰
SSDL	پاک کردن دستورالعمل	۴,۸
OEBA	انتساب ساده ازطریق انتساب دودویی	۳,۲
OLBN	عملگر منطقی وابسته به عملگر دودویی	۳,۰
OARN	عملگر ریاضی وابسته به عملگر رابطه‌ای	۲,۹
OAAN	عملگر ریاضی	۲,۷
CONS	تغییر ثابت	۲,۶
OESA	انتساب ساده ازطریق انتساب شیفت	۱,۱
OEAA	انتساب ساده ازطریق انتساب ریاضی	۰,۸

با استفاده از ابزار MZIU و عملگرهای موثر شناسایی شده و جهش‌های بکار رفته، کاربرد این جهش‌ها در زبان Ada بررسی گردید. چون زبان Ada یک زبان قوی در تعریف نوع است بسیاری از تبدیل‌ها، بین انواع جهش‌های مختلف که در زبان C امکان‌پذیر است را اجازه نمی‌دهد. بنابراین، یکسری از جهش‌ها از زیرمجموعه زبان Ada کنار گذاشته شدند که شامل جهش‌های زیر هستند.

جهش OLRN: این جایگزینی در زبان Ada تا زمانی که هر دو حالت رابطه‌ای و منطقی نتایج بولی برگردانند، مجاز است. به هر حال جای بحث است که رفتار اشتباه تولید شده با یک جایگزینی OLRN نظیر > یا =! فقط رفتار محلی تولید شده با یک جهش OLLN را تکرار می‌کند یا نه. به‌طور مثال، نیاز به استفاده جهش=! وقتی که جهش OLLN رفتار مشابه XOR را تولید می‌کند نیست. زمانی که جهش OLRN فقط یک مقایسه بین ۲ مقدار بولی را بکار می‌برد فرض می‌شود که پوشش به میزان کافی ازطریق استفاده از جهش OLLN بدست بیاید.

جهش ORLN: عملگرهای رابطه‌ای اغلب در مورد مقادیر عددی نسبت به داده بولی اجرا می‌شوند و تلاش برای استفاده یک عبارت منطقی یک مقدار در زبان Ada مجاز نیست بنابراین از این جهش در زبان Ada چشم‌پوشی خواهد شد.

جهش ORAN: یک عملگر رابطه‌ای برای برگرداندن نتایج بولی است و زبان Ada اجازه استفاده از آن را به‌جای یک عملگر محاسباتی نمی‌دهد.

جهش OLAN: استفاده از عملگرهای محاسباتی برای داده‌های بولی که کامپایلر زبان Ada آن را تشخیص می‌دهد.

جهش OARN: استفاده از عملگرهای محاسباتی برای برگرداندن نتایج عددی که زبان Ada جایگزینی عملگر رابطه‌ای به‌جای عملگر رابطه‌ای که یک مقدار بولی را برمی‌گرداند را پشتیبانی نمی‌کند.

جهش‌های OEEA, OESA, OEBA: زبان Ada ترکیب عملگرهای انتساب با سایر عملگرها را پشتیبانی نمی‌کند و این مختصرنویسی عملیات، فقط در زبان C امکان‌پذیر است و لذا از زیرمجموعه زبان Ada خارج است.

جهش OLSN: استفاده از عملگر منطقی به‌جای عملگر شیفت که نتایج بولی را برمی‌گرداند. زبان Ada این نوع از جایگزینی را پشتیبانی نمی‌کند.

جهش OLBN: استفاده از عملگرهای منطقی و بیتی به‌جای یکدیگر. در زبان Ada نیز چنین عملی ممکن نیست زیرا نوع داده و خروجی‌ها با هم متفاوتند. لذا استفاده از یک جهش OLBN، معادل استفاده از جهش OLLN است، اما در این حالت انواع داده خروجی متفاوت خواهند بود.

جهش‌های رابطه‌ای و عددی بیشترین موفقیت را در طی آزمون زبان C داشتند که منجر به نتایج مثبت شد.

جهش CONS یا تغییر ثابت با یک عملگر تکمیل می‌شود که در [31] شناسایی شد.

جهش EDT یا تغییر دامنه به‌طور موثری جهش‌های CONS را روی انواع صحیح و اعداد مثبت و منفی اجرا می‌کند و روی انواع اعشاری شامل مثبت و منفی ۵ درصد واریانس است. این نوع جهش در زبان Ada به‌صورت دستی آزمون می‌شود و موفقیت این نوع از جهش وابسته به زیرمجموعه مشتق شده است.

ISSN: 2745-3673
http://www.Rcsj.ir

جدول ۳ عملگرهای جهش را که در زبان Ada مجاز هستند نشان می‌دهد.

جدول ۳: نرخ زنده ماندن زیرمجموعه جهش در زبان Ada

جهش	ویژگی	نرخ زنده ماندن
EDT	تغییر دامنه	۱۵,۶
CONS	تغییر یک ثابت	۲,۶
ORRN	عملگر رابطه‌ای	۹,۶
OAAN	عملگر ریاضی	۲,۷
OLLN	عملگر منطقی	۶,۳
SSDL	حذف دستورالعمل	۴,۸

۳-۴- خلاصه‌ای از فرآیند آزمون جهش در دو پروژه C و Ada

اکنون می‌خواهیم با استفاده از مجموعه‌های داده آزمون، فرآیند آزمون جهش را تا۵۰ درصد بهبود دهیم. لذا باید پارامترهای آزمون نظیر کیفیت، ارزش بررسی، تمرکز روی رسیدن به اهداف پوشش و اطمینان از طراحی خوب آزمون را درنظر گیریم. اگر در تعریف داده آزمون، یک خطای ساده باشد، سبب بی‌اثر شدن آزمون نسبت به اکثریت جهش‌ها می‌گردد. براین‌اساس، مجموعه‌های داده آزمون، معیاری برای پوشش هستند و لذا می‌توان اثر بخشی اهداف آزمون و پوشش آزمون را بررسی کرد.

موثرترین جهش‌ها در بقای آزمون، جهش‌هایی هستند که مرتبط با نقاط تصمیم‌گیری درون کد هستند نظیر جهش ORRN یا عملگر رابطه‌ای جهش و جهش SSDL یا حذف دستورالعمل که سبب بهبود آزمون می‌شوند. زیرا اگر داده آزمونی ۱۰۰٪ پوشش دستورالعمل را داشته باشد به معنی اجرای همه دستورات است. دلایلی که سبب می‌شود جهش‌های SSDL زنده بمانند شامل:

۱- استفاده از آستانه تحمل بسیار وسیع در آزمون که با ورودی‌های مختلف سبب خواهد شد همه خروجی‌ها را صحیح درنظر گیرد.

۲- استفاده از عملگرهای جهش درون برنامه با پارامترهای محلی در آزمون که سبب خواهد شد سایر پارامترها در خروجی برنامه آزمون نشوند.

۳- فراخوانی‌های رویه فرزند که سبب خواهد شد قسمت‌هایی از برنامه آزمون نشوند. با این وجود بازهم جهش‌های SSDL پوشش دستورالعمل موثری دارند.

برای پروژه Ada از ۲۵ کدآیتم استفاده شد که این انتخاب براساس قابلیت عملکردی آنها است و هرکدام از آنها ۱۰۰ درصد پوشش دستورالعمل و تصمیم و MC/DC را بدست آوردند. به علت پشتیبانی محدود ابزاری برای زبان Ada، جهش‌ها به‌صورت دستی تولید می‌شدند. سایر عملگرهای جهش یا بی‌اثر بودند یا از لحاظ دستوری با زبان Ada ناسازگارند یا اثرات دیگر جهش‌ها را تکرار می‌کردند که درنهایت به ۶ زیرمجموعه عملگر کاهش یافتند. در [31]، ۵ عملگر موثر جهش برای Ada پیشنهاد شده و در [41] نشان داده است که انتخاب ۶ عملگر از ۱۱ عملگر زبان C کافی هستند و در [5] یک سری قوانین را برای انتخاب جهش توسعه داده است که یک زیرمجموعه موثر ۱۰ عملگری را انتخاب کرده است. عملگرهای جهش معرفی شده در [32] در زبان Ada، اغلب برای سیستم‌های حساس به ایمنی بوده و در این مقاله بیشتر از آنها استفاده می‌کنیم که شامل عملگرهای ABS یا درج مقدار مطلق و AOR یا جایگزینی عملگر محاسباتی و LCR یا جایگزینی اتصال منطقی و UOI یا درج عملگر یگانی است.

زیرمجموعه جهش‌های جایگزینی نظیربه‌نظیر عبارات محاسباتی و منطقی و رابطه‌ای از اساسی‌ترین عملگرها است. جهش‌های UOI و ABS مشابه جهش‌های OAAN و CONS پیشنهادی آقای آفیوت است، زیرا هردوی آنها در یک دامنه یکسان بکار رفته‌اند و از جایگزینی محاسباتی استفاده می‌کنند.

نتایج آزمون جهش در زبان Ada به این ترتیب است که ۶۵۱ جهش به‌طور دستی در ۲۵ کدآیتم نمونه تولید شدند که ۴۲ جهش یا ۶ درصد آنها توسط کامپایلر شناسایی و کشته شدند که بسیار کمتر از آن جهش‌های تولید شده به‌طور خودکار توسط MILU بودند (۱۸ درصد) و ۳۹ جهش در فرآیند آزمون زنده ماندند که ۶تای آنها به دلیل داشتن رفتار یکسان، جهش معادل تشخیص داده شدند. درنهایت ۳۳ جهش از آزمون زنده بیرون آمدند و لذا نیاز به بهبود آزمون داشتند.

ISSN: 2745-3673
http://www.Rcsj.ir

از ۲۵ کدآیتم آزمایش شده، ۸ کدآیتم امتیاز کافی برای موفقیت ۱۰۰ درصدی نداشتند و نرخ شکست، ۳۲ درصد بود که بسیار بهتر از آزمون جهش پروژه C با نرخ شکست ۵۰ درصد است. دلایل این امر شامل تعریف نوع قوی‌تر در زبان Ada، کاهش دامنه خطا، کوچکتر بودن زیرمجموعه انواع جهش‌ها و کوچکتر بودن سطوح مختلف پوشش جستجو است.

در جدول ۴، می‌بینیم که فقط ۸ کدآیتم در فرآیند آزمون جهش زبان Ada، شکست خوردند و مهم‌ترین خطا مربوط به کدآیتم A19 است که امتیاز جهش آن ۶۶/۶ درصد است. این شکست به علت یک انتخاب نادرست ضعیف از دامنه‌های ورودی، در اطراف شرایط مرزی در مجموعه آزمون است. نتایج جدول ۴ نشان می‌دهد که ارتباط شفافی بین پیچیدگی کد و نواقص آزمون وجود دارد. همچنین چون جهش‌ها در پروژه Ada به‌صورت دستی تولید و اجرا می‌شوند، کدآیتم‌های آزمون شده تمایل دارند که در اندازه‌هایی کوچکتر از پروژه C باشند.

جدول ۴: شایستگی جهش از هر کدآیتم پروژه Ada

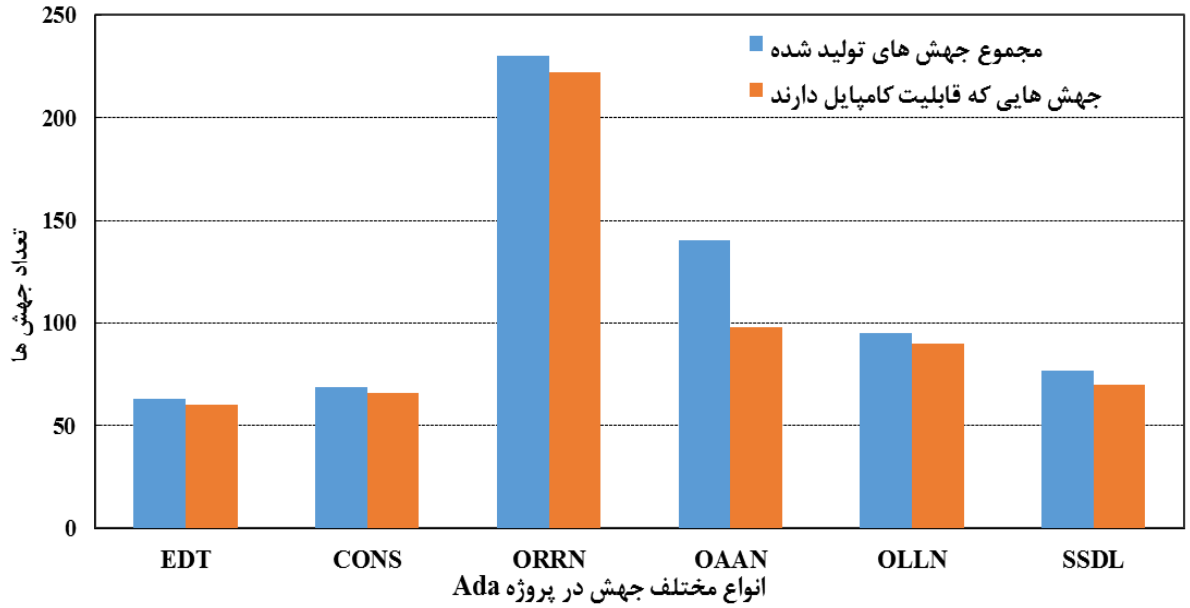
کد منبع			فاز تولید		اجرای آزمون		تحلیل		
آیتم کد	تعداد خطوط کد	پیچیدگی مداری	تعداد کل جهش‌های تولیدشده	جهش‌های نادرست نحوی	جهش‌های کشته نشده	مجموع جهش‌های مجددا آزمون شده	جهش‌های معادل	جهش‌های زنده مانده	امتیاز جهش
A1	9	1	13	0	0	13	0	0	100.00
A2	3	1	12	0	0	12	0	0	100.00
A3	9	4	38	0	4	38	0	4	89.47
A4	9	4	38	0	4	38	0	4	89.47
A5	18	3	38	10	3	28	1	2	92.59
A6	14	3	45	8	4	37	2	2	94.29
A7	11	3	18	0	0	18	0	0	100.00
A8	8	1	34	4	0	30	0	0	100.00
A9	21	2	50	2	1	48	0	1	97.92
A10	8	3	20	0	0	20	0	0	100.00
A11	6	3	27	0	1	27	0	1	96.30
A12	6	2	20	0	0	20	0	0	100.00
A13	7	2	15	4	0	11	0	0	100.00
A14	8	3	5	0	0	5	0	0	100.00
A15	6	1	19	0	0	19	0	0	100.00
A16	5	2	14	0	0	14	0	0	100.00
A17	4	2	8	0	0	8	0	0	100.00
A18	4	2	26	0	1	26	1	0	100.00
A19	4	2	18	0	6	18	0	6	66.67
A20	4	2	18	0	0	18	0	0	100.00
A21	3	2	12	0	0	12	0	0	100.00
A22	4	2	21	0	0	21	0	0	100.00
A23	18	3	38	6	1	32	1	0	100.00
A24	12	2	40	8	1	32	1	0	100.00
A25	15	6	64	0	13	64	0	13	79.69

۳-۵- انواع جهش‌های موثر:

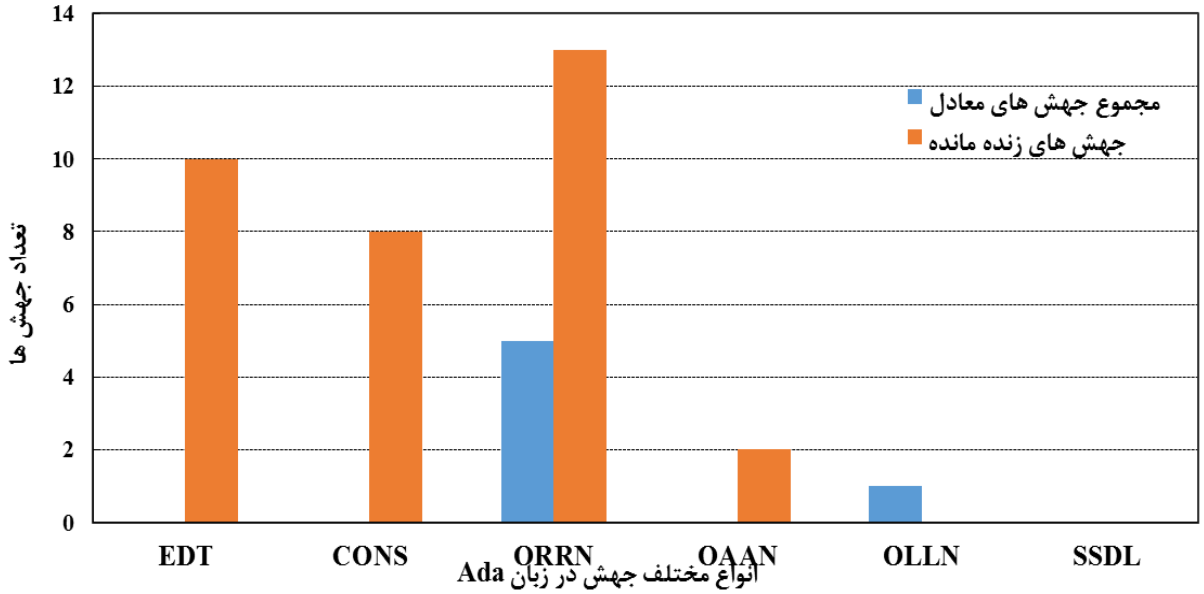
شکل ۳، تعداد جهش‌های تولید شده برای هر نوع عملگر و تعداد جهش‌های زنده مانده را در پروژه Ada نشان می‌دهد. می‌توان مشاهده کرد که زیرمجموعه‌های کاهش یافته جهش‌ها، تعداد جهش‌های کشته شده کمتری را تولید کرده است.

شکل ۴ نسبت جهش‌های معادل و جهش‌های زنده مانده در طول فرآیند آزمون را نشان می‌دهد که بسیار متفاوت با جهش‌های معادل در پروژه C است. درنتیجه زمان فرآیند آزمون Ada کمتر خواهد بود.

ISSN: 2745-3673
http://www.Rcsj.ir



شکل ۳: انواع جهش منطبق بر زبان Ada



شکل ۴: انواع جهش منطبق بر زبان Ada

در جدول ۵، تاثیر جهش‌های زنده مانده در پروژه Ada را در مقایسه با تعداد کل انواع جهش‌های تزریق شده نشان می‌دهد. که فقط ۴ جهش CONS، EDT، OAAN و ORRN از ۶ جهش استفاده شده در مجموعه آزمون زنده مانده‌اند. این نتایج نشان می‌دهد که SSDL به‌علت پوشش سطح دستورالعمل، بی‌اثر است. کارایی ضعیف، جهش OLLN با یافته‌های موجود در [9] نیز هم‌خوانی دارد زیرا عملگر منطقی، پوشش اندکی دارد. به‌عنوان مثال MC/DC سبب کاهش روی زیرمجموعه‌های بیشتر جهش شده و جهش‌های انتخابی را کاهش می‌دهد [32].

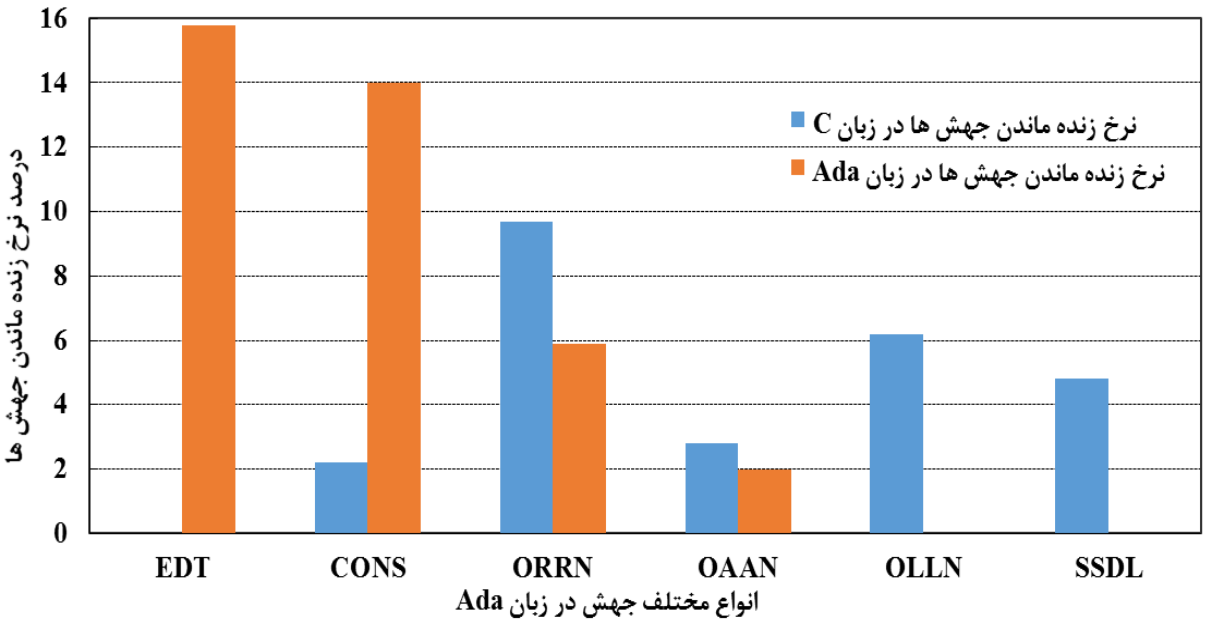
ISSN: 2745-3673
http://www.Rcsj.ir

جدول۵: نرخ زنده ماندن زیرمجموعه جهش‌ها در پروژه Ada

جهش	ویژگی	نرخ زنده ماندن
EDT	تغییر دامنه	۱۵,۶
CONS	تغییر یک ثابت	۱۴,۰
ORRN	عملگر رابطه‌ای	۵,۸
OAAN	عملگر ریاضی	۲,۰
OLLN	عملگر منطقی	۰,۰
SSDL	حذف دستورالعمل	۰,۰

چون زیرمجموعه عملگرهای جهش شامل جهش‌های پایه است، در دو پروژه اغلب خطاهای وارد شده یکسان بودند. به‌عنوان مثال، جهش ORRN در هر دو پروژه C و Ada، پنج تغییر را ایجاد و تزریق می‌کند. تنها تفاوت اصلی بین این پروژه‌ها، پوشش سطح آزمون است. در آزمون پروژه Ada هیچ یک از جهش‌های OLLN و SSDL زنده نماندند و در هر دو پروژه جهش‌های OAAN به‌طور مشابه‌ای موثرند. از روی جهش‌های زنده مانده در هر دو پروژه، می‌توان با بهبود داده آزمون، آزمون در هر دو پروژه را بهبود داد.

شکل ۵ نرخ درصد زنده ماندن هر نوع جهش را نشان می‌دهد. جهش‌های CONS و EDT با دستکاری عددی و جایگزینی رابطه موفق‌ترین و جهش‌های SSDL و OLLN ناموفق‌ترین بودند که علت آن تفاوت در سطوح پوشش آزمون بین دو پروژه است و لذا می‌توان زیرمجموعه جهش‌ها را بیشتر کاهش داد. کارآمد نبودن جهش‌های OLLN دربرابر پوشش MC/DC قبلا در [9] نیز اشاره شده است. یک نقص کوچک، تاثیر قابل توجهی روی امتیاز جهش دارد و علت آن محدود بودن جهش‌های بکار رفته است. نتایج بدست آمده از هر دو پروژه نشان می‌دهد که علت اصلی جهش‌های زنده مانده، پیچیدگی کد و افزایش اندازه کد است.



شکل ۵: انواع جهش منطبق بر زبان Ada

۳-۶- ارزیابی نواقص در داده آزمون:

دلایل اساسی برای زنده ماندن جهش‌ها در هر دو پروژه اغلب یکسان و ثابت هستند. همه دستورالعمل‌های صنعتی موجود و مجموعه‌های داده آزمون باید منطبق بر نیازهای DO-178B باشند و در هر دو پروژه فعالیت جزء آزمون توسط یک شخص ثالث اجرا می‌شود و یک نمونه از مجموعه‌های داده آزمون به‌عنوان یک بازنگری دستی توسط AEC انجام می‌شود. بنابراین ازنظر صنعتی، آزمون جهش نه تنها شواهدی را برای کارایی آزمون فراهم می‌کند، بلکه سبب افزایش کیفیت کار بدون نیاز به کدمنبع نیز خواهد شد.

ISSN: 2745-3673
http://www.Rcsj.ir

اکثر نتایج فرآیند جهش، بیانگر بهبودهایی در طراحی داده آزمون است. مستندات بهتر از ابزار آزمون و استانداردهای تائید می‌تواند مانع از شکست‌ها شود. منشا برخی از مشکلات، درون مجموعه‌های داده آزمون و یا فرآیندهای برنامه‌نویسی است. تحلیل جهش‌های زنده، همواره سبب یک درک کامل از اهداف داده آزمون می‌شود. علت ناقص بودن مجموعه‌های داده آزمون، عدم فهم از نظرات و جزئیات نظیر هدف آزمون، چگونگی پوشش شاخه، انتخاب مقدار ورودی و غیره است. همچنین علت اصلی برخی شکست آزمون‌ها، عدم همخوانی مجموعه‌های داده آزمون با استانداردها است.تحلیل دستی پیدا کردن این یافته‌ها ممکن است دشوار باشد اما با استفاده از آزمون جهش دقیقا می‌توان محل بهبود دادن مجموعه آزمون را یافت. جدول ۶ انواع آزمون ناقص شناسایی شده و علل آنها و راه‌های کشف آنها را طبقه‌بندی کرده است.

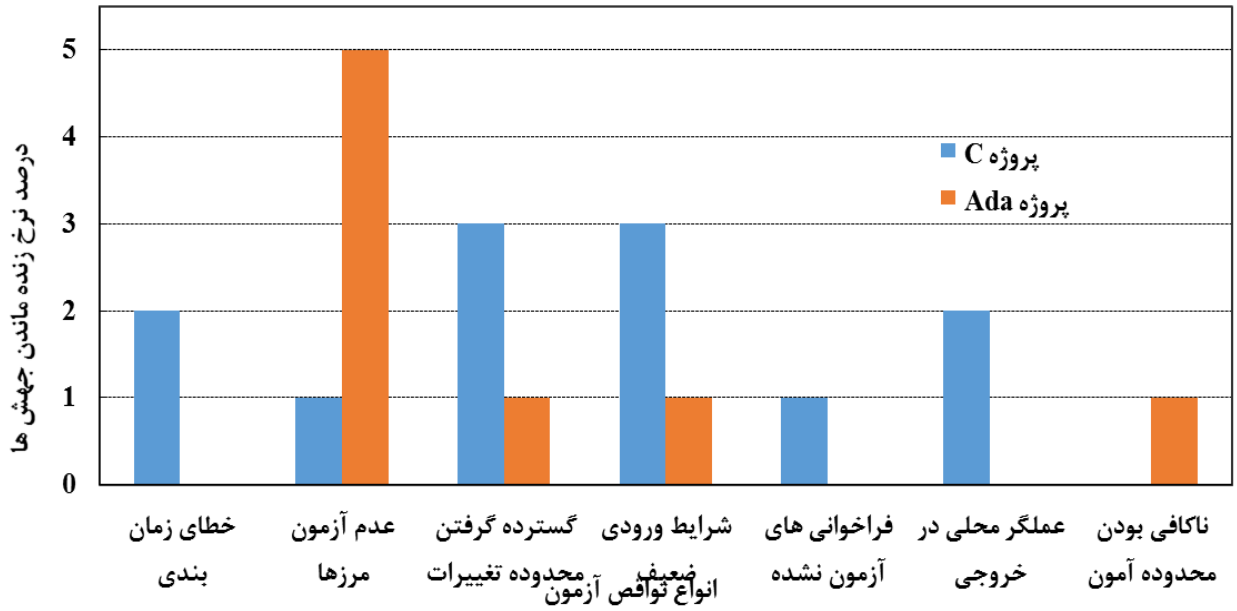
جدول۶: طبقه‌بندی نواقص آزمون در هر دو پروژه

نقص آزمون	رخداد در مجموعه آزمون	انواع جهش شناخته شده برای این مورد	تعریف
خطا در موارد آزمون	C4 , C11	OE*A , OLRN , SSDL	یک آزمون که یا با نحو زبان یا ازنظر مهندسی قابل قبول نباشد و در پوشش باتوجه به ساختار آزمون نقص دارد مانند داشتن پیچیدگی بالا از یک مورد آزمون که یا اولویت‌دار است یا خطاهای نحوی دارد.
شرایط محدوده‌ای کاملا آزمون نشود	C6 , A3 , A4 , A11 , A19 , A25	CONS , EDT , ORRN	یعنی یک یا دو طرف حد یا تساوی و شرایط محدوده‌ای آنها کاملا آزمون نشود.
دامنه تغییرات آزمون وسیع باشد	C8 , C16 , C20 , A5	CONS , OAAN , OARN , OLRN , ORRN , SSDL	اگر دامنه تغییرات خیلی بزرگ و وسیع باشد، بررسی خروجی و تشخیص بروزرسانی داده پیشبینی شده یا یک مورد آزمون مشکل خواهد بود.
انتخاب ضعیف داده آزمون	C10 , C19 , C21 , A9	OEAA , OEBA , ORRN	نمونه‌هایی که انتخاب ضعیفی از داده آزمون‌هایی که روی جهش‌ها رفتاری معادل با کداصلی برنامه دارند و خروجی را تغییر نمی‌دهند که با ورودی‌های بیشتر آزمون داده آزمون‌ها قوی‌تر شده و می‌توانند جهش‌های بیشتری را بکشد.
فراخوانی رویه‌های فرزند که توسط والدشون آزمون نمی‌شوند.	C14	SSDL	اگر رویه فرزند یک مورد کد را فراخوانی کند و یک حالت جدید را ایجاد و برگرداند، چون در این مرحله از معماری چنین عملیاتی آزمون نمی‌شوند، سبب نقص در آزمون می‌گردد.
عدم آزمون خروجی با عملیات صحیح و نشانه‌های محلی	C15 , C16	CONS , OAAN , OLRN , SSDL	شامل اجرا شدن عملیات با استفاده از پارامترهای محلی و غیرقابل مشاهده در آزمون که در کدآیتم خروجی، آزمون نمی‌شوند.
عدم کافی بودن محدوده داده آزمون	A6	OAAN , ORRN	یعنی شرایط داده آزمون بررسی و اجرای شرایط محدوده‌ای و مرزی کافی و اطمینان بخش نباشد. برای کشتن جهش‌ها و آزمون کننده شکست خورده و برای محاسبه خطاهای گرد شده صحیح و دقیق یا دامنه تغییرات محدوده کافی نباشد.

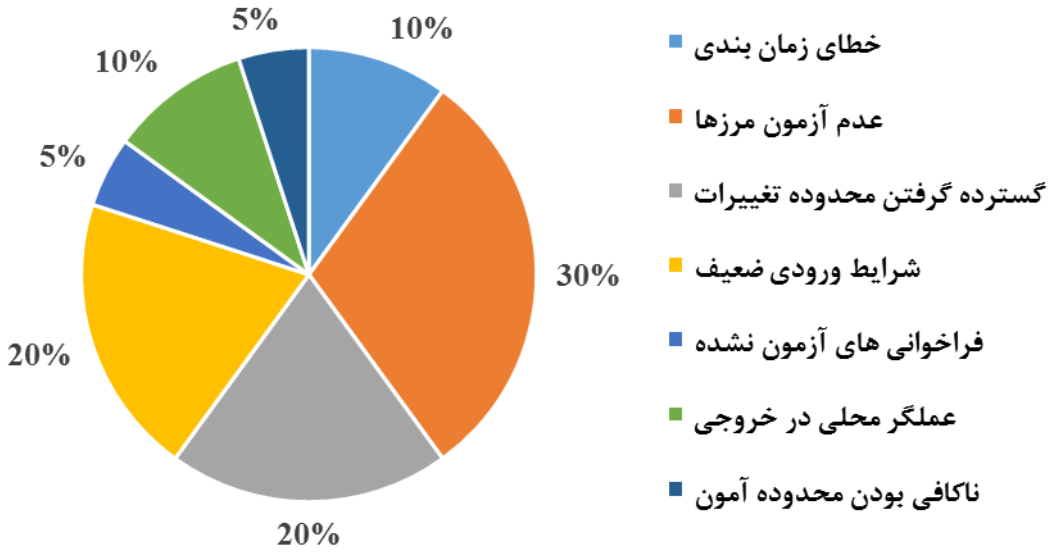
شکل ۶ توزیع نواقص موجود در آزمون را در طول هر دو پروژه نشان می‌دهد.

شکل ۷ توزیع ترکیبی نواقص آزمون را در طول هر دو پروژه نشان می‌دهد. شکست‌های غالب وابسته به انتخاب داده آزمون‌ها است و عدم آگاهی از رفتار داده آزمون‌ها به دلیل نبود اطلاعات و نیازمندی‌ها از برنامه است.

ISSN: 2745-3673
http://www.Rcsj.ir



شکل ۶: توزیع نواقص آزمون در طول هر دو پروژه



شکل ۷: توزیع ترکیبی نواقص آزمون در طول هر دو پروژه

عواقب خطاهای داده آزمون مربوط به شرایط مرزی به‌عنوان مثال تغییر => با >، ممکن است سبب ظاهر شدن در کد به‌صورت یک خطای کمتر از ۱ بیت گردد. علت این دسته از مشکلات اغلب مربوط به دسترسی به عملیات محلی یا تعریف نیازمندی‌های سطح پایین یا تصمیمات طراحان مرتبط است. انواع شکست‌های شناسایی شده و جهش‌هایی که مکان‌یابی آنها به ما در ایجاد یک درک اینکه کدام جهش‌ها، خطاهای بیشتری را توسط آزمون کننده‌ها آشکار می‌کنند، آزمون کننده‌ها را مجبور به استفاده از تکنیک‌های قوی‌تری کرده است. روش ما برای ایجاد جهش، با روش Offutt همخوانی دارد به‌طوری‌که عملگرهای جهش، خطاهایی را که اغلب توسط برنامه‌نویسان ایجاد می‌شود را شبیه‌سازی می‌کنند و همچنین خطاهایی را که مهندسان آزمون را مجبور به استفاده از یک روش سخت‌تر می‌کند را شامل می‌شود [32]. اگرچه هدف اصلی استفاده از آزمون جهش فهم مولفه‌های مورد نیاز در طراحی داده آزمون است اما بعضی از مشکلات در خلال آزمون جهش پیدا شدند که وابسته به اقدامات اصلاحی و در محدوده آزمون شبکه بودند. بنابراین کیفیت ورودی داده آزمون، تاثیر عمده‌ای روی اطمینان آزمون دارد.

ISSN: 2745-3673
http://www.Rcsj.ir

۳-۷- ارتباط بین مشخصات برنامه و تولید جهش و نرخ زنده ماندن

برخی مشخصات برنامه نظیر تعداد خطوط و پیچیدگی کد و پیچیدگی مداری با هزینه‌های توسعه و ارزیابی آزمون و نیاز به تعریف مجموعه‌های داده آزمون پیچیده و طولانی شدن فرآیند آزمون ارتباط دارند. پیچیدگی مداری، تعداد مسیرهای مستقل در یک برنامه است که هرچه افزایش یابد، آزمون و نگهداری عملکر صحیح واحدهای این برنامه مشکل‌تر خواهد بود. همچنین اغلب بین اندازه یک برنامه و پیچیدگی مداری یک ارتباط مستقیم وجود دارد و با افزایش پیچیدگی کد، اغلب نواقص آزمون نیز افزایش می‌یابد. در [25] و [40] مفهوم پیچیدگی مداری را توسعه دادند که با تکرار خطاهای کد مرتبط است. جهش‌های مرتبط با پیچیدگی مداری اغلب به نقاط تصمیم و مسیرهای کد مرتبط‌اند و هرچه این جهش‌ها موفق‌تر باشند؛ احتمالا روی رابطه با پیچیدگی مداری تاثیر بیشتری دارند.

در پروژه C، هرچه پیچیدگی اندازه و کدآیتم‌ها ها افزایش یابد تعداد جهش‌های تولید شده و نرخ زنده ماندن جهش‌ها نیز افزایش می‌یابد. لازم است برای تاثیر شکست‌های خارج از محدوده مجموعه آزمون برای کدآیتم C8 ملاحظات ویژه‌ای درنظر گرفته شود زیرا یک خطای کوچک درون آزمون منجر به کسب امتیاز جهش فقط ۳۷ می‌شود و یک چنین شکستی تاثیر عمده‌ای روی پوشش آزمون دارد. اما نمونه کدآیتم C8 یک استثنا بوده و روند کلی نتایج را منحرف می‌کند اما امتیاز جهش ۸۱ درصد از کل مجموعه نمونه‌های پروژه C که با نواقص شناسایی شدند، بالاتر از ۹۴ شد.

هر جهش زنده کیفیت آزمون را تعیین می‌کند و سبب بهبود داده آزمون مربوط آن می‌شود. اما کدآیتم‌های کوچک‌تر، اغلب وزن کمتری دارند که با برداشتن این کدآیتم‌های کوچک از تحلیل، توانایی درک بهتری از رابطه بین پیچیدگی پیدا می‌کنیم.

امتیاز پیچیدگی مداری ۷۵ درصد کدآیتم‌ها در پروژه Ada که نواقص آزمون را شناسایی کردند، از ۳ به بالا است که Waston و MaCab بالاترین حد برای پیچیدگی مداری کدآیتم‌ها را ۱۰ توصیه کردند که این حد با توجه به مزایای عملیاتی نظیر تجربه کارکنان طراحی رسمی، زبان برنامه‌نویسی مدرن و یک نقشه آزمون جامع می‌تواند تا ۱۵ گسترش یابد. لذا یک ارتباط ثابت و مسقیم بین پیچیدگی و اندازه کد و تولید جهش و نرخ زنده ماندن جهش‌ها وجود دارد.

اثبات رابطه بین نقص آزمون یا نرخ‌های زنده ماندن و پیچیدگی کد و اندازه کد از لحاظ آماری بسیار مهم است و نیاز به آزمون‌های بیشتر و انتخاب نقاط تصمیم مهم‌تر و انتخاب داده آزمون مطمئن‌تر دارد که خارج از محدوده این مقاله است.

یک ارتباط بالقوه بین پیچیدگی کد و نرخ زنده ماندن جهش می‌تواند به‌دلیل تعداد زیاد شاخه‌های انشعابی در کد و جهش‌های رابطه‌ای باشد. جهش‌های ORRN سبب افزایش نرخ زنده ماندن جهش‌ها در هر دو پروژه شده و جهش‌های OLLN تاثیر کمتری در نرخ زنده ماندن جهش‌ها در هر دو پروژه دارد. در [18,27,36]ها نیز به رابطه بین نرخ زنده ماندن و پیچیدگی کد یا LOC و پیچیدگی مداری یا CC اشاره شده است و نشان داده‌اند که افزایش یکی از CC یا LOC سبب افزایش دیگری خواهد شد لذا یک رابطه‌ای خطی بین آنها وجود دارد.

۳-۸- مقایسه با فرآیند بازنگری دستی

اغلب کدآیتم‌های انتخاب شده برای آزمون مربوط به عملکرد سیستم‌های‌عامل بودند. کد یک سیستم‌عامل مرتبط با سخت‌افزار و ورودی و خروجی و واسطها بوده و اطلاعات خام را در برنامه‌های کاربردی قرار می‌دهد. اغلب عملکردهای سیستم‌عامل اجرای اطلاعات پایه است و لذا جهش‌های محدودی را می‌توان در حوزه سیستم‌عامل گرفت. زیرمجموعه جهش‌های بکار رفته در خلال پروژه Ada شامل SSDL، ORRN، OLLN، OAAN، CONS/EDT است که در جدول ۷ کدآیتم‌های اضافه شده به‌عنوان آیتم‌های R1 تا R3 نشان داده شده است. این جدول نشان می‌دهد که امتیاز جهش، برای زیرمجموعه آزمون کافی نیست.

جدول ۷: کدآیتم‌های اضافه به پروژه Ada

کدآیتم	تعدادخطوط برنامه	پیچیدگی مداری	جهش‌های کاربردی	جهش‌های زنده مانده
R1	۲۷۶	۴	۲۷	۴
R2	۹	۳	۱۳	۲
R3	۵۲	۲۴	۸۸	۲

چون جهش‌های استفاده شده براساس انتخاب آگاهانه برای جلوگیری از تولید جهش‌های معادل و مرده انتخاب شده‌اند و شامل همه جهش‌ها نمی‌شوند، حجم کار کمتری در تولید زیرمجموعه جهش‌های دستی وجود دارد و یک مهندس نرم‌افزار با یک درک مناسب از کد

ISSN: 2745-3673
http://www.Rcsj.ir

می‌تواند ارزیابی اولیه و تولید جهش و هماهنگ‌سازی جهش و ارزیابی رفتار معادل بودن جهش را انجام دهد. زمان صرف شده برای شناسایی صحیح انواع جهش‌ها از قبل به‌طور قابل توجهی زمان تحلیل و اجرای آزمون را کاهش داد.

خلاصه‌ای از نتایج آزمون جهش به‌شرح زیر است:

۱- هر سه مجموعه نمونه آزمون شده با استفاده از زیرمجموعه در فرآیند جهش با مشاهده نواقص شکست خوردند.

۲- کدآیتم R1 مسوول انتقال اطلاعات بوده و دو مشکل جداگانه و عمده پوشش دارد که شامل:

الف- شمارشگر حلقه به‌طور افزایشی اطلاعات نوشته شده را ازطریق یک اتصال سریال به خارج ارسال می‌کند و از تعدادی محل براساس تعریف نیازمندی‌ها استفاده می‌کند و لذا فرض شده که داده آزمون‌ها هیچ وقت فراتر از اندازه حلقه قرار نمی‌گیرند و هیچ عملیات نوشتن اضافی رخ نمی‌دهد.

ب- اگر توسط عملگر جهش یک داده روی داده دیگری نوشته شود و جهش در این قسمت کشته نشود، سبب شده تا پاکسازی مکان نیز کشف نشود و درنتیجه ازطریق آزمون جهش نمی‌توان نشان داد که آیا الگوهای اطلاعاتی فردی به‌طور صحیح انتقال یافته‌اند یا نه.

۳- کدآیتم R2 از یک عملگر حلقه ساده تشکیل شده و از تعدادی محل‌های سخت‌افزاری ثابت استفاده می‌کند. هر جهش زنده مربوط به ساختار این حلقه شامل عدم دسترسی نمونه آزمون به محل‌های سخت‌افزاری است و هیچ آزمونی برای اطمینان از اینکه شاخص حلقه به بیشترین مقدار خود رسیده و از کد خارج می‌شویم، وجود ندارد. با این وجود این سطح از استحکام درون کدآیتم مطلوب است.

۳- کدآیتم R3 در آزمون جهش شکست خورده است زیرا نشانه‌های پاک شده از کد را پیدا نکرده است.

در خلال آزمون جهش هیچکدام از مشکلات فوق توسط بازنگری دستی شناسایی نشدند. این نشان می‌دهد که هر خطا یا خیلی مبهم بوده که نتوانسته با بررسی دستی شناسایی شود یا فرآیند بررسی نیاز به تقویت دارد.

مشکل کدآیتم C11 در مجموعه آزمون بوده که علت آن اضافه شدن هیچ نوع مقدار خاصی از خارج بوده است. آزمون جهش ثابت کرد که هرچه دستورات آزمون پیچیده‌تر شوند، پوشش کدآیتم‌ها نیز مشکل‌تر خواهند شد که این مطلب را در جدول ۶ برای شکست‌های C4 و C11 نیز می‌توان دید. همچنین این موارد را در خلال آزمون جهش کدآیتم‌های A9، C20، C8 نیز می‌توان مشاهده کرد.

جدول ۸، موفق‌ترین جهش‌های بهبود بررسی دستی را نشان می‌دهد. در این جدول نیز می‌توان ناکارآمدی جهش OLLN را مشاهده کرد. بنابراین با استفاده از آزمون جهش و بررسی داده آزمون‌ها و جهش‌ها می‌توان پوشش مناسبی از کد را فراهم کرد.

جدول ۸: بررسی دستی زیرمجموعه موثر عملگرهای جهش

نوع جهش	جهش‌های‌زنده‌مانده در فاز خودکار Ada	جهش‌های‌زنده‌مانده در فاز دستی
CONS/EDT	بله	بله
OAAN	بله	خیر
OLLN	خیر	خیر
ORRN	بله	بله
SSDL	خیر	بله

۳-۹- بازخورد صنعتی

فرآیندهایی که در این مقاله استفاده شده است، توانایی اجرای نقاط بررسی اطمینان و پشتیبانی از کیفیت مهندسی را دارند. نگرانی‌ها شامل حفظ استقلال درونی و نظم وارسی در استفاده از آزمون جهش است. اغلب استقلال آزمون جهش که یک تکنیک آزمون جعبه سفید است توسط آزمون جعبه سیاه در یک سطح مولفه نگهداری می‌شود. در این مقاله، فرآیند جهش را در یک کدآیتم با اندازه متوسط C با ۲۰ خط و پیچیدگی ۳,۷ و در یک کدآیتم Ada با ۹ خط و پیچیدگی ۲,۵ اجرا کردیم که جهش می‌تواند مجددا از نو اجرا شده و تقریبا در یک قالب زمانی یکسان تحلیل شود که نیازمند یک بازنگری کامل دستی است.

فرآیند آزمون جهش در پروژه C، نیاز به چندین ماه و چندین فاز در خلال تصفیه و تکرار و ترکیب آنها با یکدیگر دارد. تولید خودکار جهش سبب صرفه‌جویی در زمان شده اما سپس تعداد زیادی از جهش‌ها نیازمند تحلیل خواهند بود.

فرآیند آزمون جهش در پروژه Ada، بیشتر تصفیه شد و ساخت جهش و اجرای آزمون و نتایج تولید شده به‌صورت خودکار و بدون دخالت دست انجام شد. تولید جهش به‌صورت دستی انجام شد، اما چون روی زیرمجموعه‌های عملگرهای جهش کوچک و ثابت تمرکز داشتیم، سربار تولید جهش‌های غیرضروری کاهش یافت.

ISSN: 2745-3673
http://www.Rcsj.ir

۳-۱۰- تهدیداتی برای اعتبار آزمون

اعتبار یک آزمون شامل اعتبار داخلی و خارجی و ساختاری است

در این مقاله، برای حفظ اعتبار داخلی از کدهایی با ساختار متفاوت و قابلیت‌های متنوع و با تعداد خطوط مختلف و سطوح پیچیدگی متفاوت به‌منظور به حداکثر رساندن تعداد انواع جهش‌ها در هر دو پروژه نرم‌افزاری Ada و C استفاده کردیم که اندازه ساختار کد از ۳ تا ۴۶ خط دستورالعمل و سطوح پیچیدگی مداری از دامنه ۱ تا ۱۱ است. با وجود تفاوت در زبان برنامه‌نویسی، عملیات سطح پایین در هر دو پروژه یکسان است. برای پروژه C، ۴۶ نوع جهش در ۲۲ کدآیتم استفاده شد که فقط ۵ تا از این جهش‌ها برای ۲۵ کدآیتم در پروژه Ada استفاده شد که دلیل این محدودیت تعریف نوع قوی در زبان Ada است و معادل نبودن آن با دستورات زبان C است.

در این مقاله، برای حفظ اعتبار خارجی از دو سیستم نرم‌افزاری صنعتی حساس به ایمنی و هوایی با حفظ نیازمندی‌های DO-178B برای اطمینان از مراحل C و A استفاده شد که کنترل اجرایی و عملکرد نظارتی آن برعهده یک موتور aero است.

همچنین در این مقاله برای حفظ اعتبار خارجی از زیرمجموعه‌های زبانی با یکپارچگی بالا در SPARK Ada و MISRA C در زمینه‌های دفاعی و خودرو نیز استفاده شد که معیار ساختار پوششی آن نزدیک به اهداف پوشش در استانداردهای [16,17] است. همچنین سازمان هوافضا درحال بروزرسانی DO-178B به DO-178C است تا پوشش دستورالعمل و MC/DC بیشتر رعایت شود.

در این مقاله، برای حفظ اعتبار ساختاری از پارامترها اندازه یا تعداد خطوط برنامه یا LOC و پیچیدگی مداری برنامه که توسط سیستم اندازه‌گیری AEC محدود شده است استفاده شده است. این پارامترها معیارهای خوبی برای مقایسه نرخ جهش‌های زنده مانده در طول انجام کدآیتم‌های پروژه‌های Ada و C هستند. پارامترهای دیگر نظیر کفایت جهش و امتیاز جهش نیز برای بررسی کیفیت آزمون استفاده شدند.

ISSN: 2745-3673
http://www.Rcsj.ir

نتیجه‌گیری

در این مقاله آزمون جهش را بررسی کرده و کمبودها و نواقص اساسی آن را به‌صورت دستی شناسایی کرده تا اهمیت و ارزش فرآیند جهش افزایش یابد. همچنین نشان دادیم که ازطریق آزمون جهش می‌توان سازگاری و کیفیت آزمون را بررسی کرد تا آزمون جهش بتواند جایگزین بررسی دستی شود و قابلیت درک و بررسی را افزایش دهد. همچنین نشان دادیم که معیارهای پوشش برای شناسایی مسائل فوق کافی نیست. به‌طوری‌که اکثر مقالات در زمینه آزمون جهش، اغلب روی اهداف پوشش رضایت‌بخش تمرکز کرده‌اند و کمتر روی تولید و طراحی خوب آزمون تمرکز دارند. همچنین ارزش معیار پوشش سطح دستورالعمل را بررسی کرده و نشان دادیم که این معیار نمی‌تواند به تنهایی حداقل الزامات DO-178B را برای یک نرم‌افزار در سطح A فراهم کند. همچنین نشان دادیم که چگونه آزمون جهش می‌تواند در روش‌های پوشش سنتی که ممکن است شکست بخورد مفید واقع گردد. همچنین ارتباط بین ویژگی‌های برنامه و نرخ خطاهای زنده مانده در برنامه را نیز بررسی کردیم. درنتیجه می‌توان آزمون را توسعه داد و با تقویت داده آزمون‌ها خطاهای پنهان را یافت و استانداردهایی نظیر DO-178B را توسعه داد تا آزمون جهش در نرم‌افزارهای حساس به ایمنی و بحرانی بیشتر بکار رود. این مقاله همچنین تفاوت بین سطوح پوشش را بررسی کرده و تاثیر آنها را روی کیفیت آزمون نشان می‌دهد. تجزیه و تحلیل نتایج آزمون‌ها و تعیین رفتار عملگرهای جهش و یافتن جهش‌های معادل نیاز به یک سربار دستی دارد. این مقاله همچنین ساخت جهش و مراحل آزمون را به‌صورت خودکار و بدون نیاز به هیچ‌گونه دخالت دستی انجام می‌دهد اما تاکنون در زبان Ada تولید جهش فقط بصورت دستی انجام شده است. همچنین نیاز است که بهبودهای آزمون ازطریق آزمون جهش به سمت افزایش یادگیری میل نماید و الگوهای داده آزمون آنقدر قوی باشند تا قابلیت استفاده مجدد در سراسر برنامه‌های کاربردی را داشته باشند و سعی در کاهش نواقص برنامه داشته باشند.

ISSN: 2745-3673
http://www.Rcsj.ir

منابع و مراجع

[1] H. Agrawal, R.A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E.W.Krauser, R.J. Martin, A.P. Mathur, and E. Spafford, "Design of Mutant Operators for the C Programming Language," Technical Report SERC-TR-41P, Purdue Univ., West Lafayette, Ind., Mar. 1989.

[2] J.H. Andrews, L.C. Briand, and Y. Labiche, "Is Mutation an Appropriate Tool for Testing Experiments?" Proc. IEEE Int'l Conf. Software Eng., pp. 402-411, 2005.

[3] J.H. Andrews, L.C. Briand, and Y. Labiche, A.S. Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," IEEE Trans. Software Eng., vol. 32, no. 8, pp. 608-624, Aug. 2006.

[4] D. Baldwin and F.G. Sayward, "Heuristics for Determining Equivalence of Program Mutations," Research Report 276, Yale Univ., New Haven, Conn., 1979.

[5] E. Barbosa, J.C. Maldonado, and A. Vincenzi, "Toward the Determination of Sufficient Mutant Operators for C," Software Testing, Verification, and Reliability, vol. 11, pp. 113-136, 2001.

[6] J. Barnes, High Integrity Software: The SPARK Approach to Safety and Security. Addison-Wesley, 2003.

[7] R. Butler and G. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," IEEE Trans. Software Eng., vol. 19, no. 1, pp. 3-12, Jan. 1993.

[8] J.J. Chilenski and S.P. Miller, "Applicability of Modified Condition/Decision Coverage to Software Testing," Software Eng. J., vol. 9, no. 5, pp. 193-200, 1994.

[9] J.J. Chilenski, "An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion," Report DOT/ FAA/AR-01/18, Office of Aviation Research, Washington, D.C., Apr. 2001.

[10] H. Do and G.E Rothermel, "On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques," IEEE Trans. Software Eng., vol. 32, no. 9, pp. 733-752, Aug. 2006.

[11] D. Daniels, R. Myers, and A. Hilton, "White Box Software Development," Proc. 11th Safety-Critical Systems Symp., Feb. 2003.

[12] M. Daran and P. The´venod-Fosse, "Software Error Analysis: A Real Case Study Involving Real Faults and Mutations," ACM SIGSOFT Software Eng. Notes, vol. 21, no. 3, pp. 158-177, May 1996.

[13] R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Hints on Test Data Selection: Help for the Practical Programmer," Computer, vol. 11, no. 4, pp. 34-41, Apr. 1978.

[14] K. Hayhurst, D.S. Veerhusen, J.J. Chilenski, and L.K. Rierson, "A Practical Tutorial Decision Coverage," NASA Report, NASA/TM- 2001-210876, 2001.

[15] C.R.M. Hierons, M. Harman, and S. Danicic, "Using Program Slicing to Assist in the Detection of Equivalent Mutants," Software Testing, Verification, and Reliability, vol. 9, no. 4, pp. 233-262, Dec. 1999.

[16] Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, IEC 61508, Int'l Electrotechnical Commission, Mar. 2010.

[17] ISO 26262: Road Vehicles: Functional Safety, Int'l Organization for Standardization, June 2011.

[18] G. Jay, J.E. Hale, R.K. Smith, D.P. Hale, N.A. Kraft, and C. Ward, "Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship," J. Software Eng. and Applications, vol. 3, no. 2, pp. 137-143, 2009.

[19] Y. Jia and M. Harman, "Higher Order Mutation Testing," Information and Software Technology, vol. 51, no. 10, pp. 1379- 1393, 2009.

[20] Y. Jia and M. Harman, "MILU: A Customizable, RuntimeOptimized Higher Order Mutation Testing Tool for the Full C Language," Proc. Third Testing Academia and Industry Conf.Practice and Research Techniques, Aug. 2008.

[21] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing, Software Engineering," IEEE Trans. Software Eng., vol. 37, no. 5, pp. 649-678, Sept./Oct. 2011.

ISSN: 2745-3673
http://www.Rcsj.ir

[22] J.A. Jones and M.J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage," IEEE Trans. Software Eng., vol. 29, no. 3, pp. 195-209, Mar. 2003.
[23] B. Littlewood and L. Strigini, "Validation of Ultrahigh Dependability for Software-Based Systems," Comm. ACM, vol. 36, no. 11, pp. 69-80, 1993.
[24] T.J. McCabe, "A Complexity Measure," IEEE Trans. Software Eng.,vol. 2, no. 4, pp. 308-320, Dec. 1976.
[25] T.J. McCabe and A.H. Watson, "Combining Comprehension and Testing in Object-Oriented Development," Object Magazine, vol. 4, pp. 63-64, Mar./Apr. 1994.
[26] J.A. McDermid, Software Engineer's Reference Book. ButterworthHeinemann Newton, 1991.
[27] V.D. Meulen and M.A. Revilla, "Correlations between Internal Software Metrics and Software Dependability in a Large Population of Small C/C++ Programs," Proc. 18th IEEE Int'l Symp. Reliability, Nov. 2007.
[28] MISRA, "Guidelines for the Use of the C Language in Critical Systems," Oct. 2004.
[29] P.R. Muessig, "Cost vs Credibility: How Much V&V Is Enough?" Naval Air Warfare Center, Weapons Division, China Lake, Calif., 2001.
[30] G.J. Myers, The Art of Software Testing. John Wiley & Sons, 2004.
[31] A.J. Offutt, J. Voas, and J. Payne, "Mutation Operators for Ada," Technical Report ISSE-TR-96-09, Information and Software Systems Eng., George Mason Univ., 1996.
[32] A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch, and C. Xapf, "An Experimental Determination of Sufficient Mutant Operators," ACM Trans. Software Eng. and Methodology, vol. 5, no. 2, pp. 99- 118, Apr. 1996.
[33] A.J. Offutt and J. Pan, "Automatically Detecting Equivalent Mutants and Infeasible Paths," Software Testing, Verification, and Reliability, vol. 7, no. 3, pp. 165-192, Sept. 1997.
[34] A.J. Offutt and R.H. Untch, "Mutation 2000: Uniting the Orthogonal," Proc. Mutation 2000: Mutation Testing in the Twentieth and the Twenty-First Centuries, 2000.
[35] "DO-178B Software Considerations in Airborne Systems and Equipment Certification," RTCA, Washington, D.C., 1992.
[36] M. Shepperd, "A Critique of Cyclomatic Complexity as a Software Metric," Software Eng. J., vol. 3, no. 2, pp. 30-36, Mar. 1988.
[37] "Aerospace Recommended Practice 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems," Soc. Of Automotive Eng. (SAE), Nov. 1996.
[38] "ARP4761—Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Soc. of Automotive Eng. (SAE), 1996.
[39] M. Umar, "An Evaluation of Mutation Operators for Equivalent Mutants," master's thesis, King's College, London, 2006.
[40] A.H. Watson and T.J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," Computer Systems Laboratory, Nat'l Inst. of Standards and Technology, Gaithersburg, Md., Sept. 1996.
[41] W.E. Wong, J.C. Maldonado, M.E. Delamaro, and S.R.S Souza, "A Comparison of Selective Mutation in C and Fortran," Proc. Workshop Validation and Testing of Operational Systems Project, Jan. 1997.